Exceptions

Logistics

- Project
  - Part 2 (water) due Sunday, Oct 16th
  - Return & feedback
  - Part 3 (block) due Sunday, Oct 30
- Questions?

Project

- Part 2
  - Intuitive output
  - Solver loop must be generic
  - Very small main
  - Documentation!!!
  - Can’t lose points

- Part 3: Block Puzzle
  - What I’ll be looking for:
    - Functionality (10)
    - Use of framework (5)
    - Code Style (5)
    - Memoization (5)

Exam

- Exam 2
  - Thursday, October 27.

  - Review Session
    - Tuesday, Oct 25th / 9-10am (70-3435)
    - Wednesday, Oct 26th / 8-9pm (70-3445)

Exam 2

- What it will cover
  - C++ classes
    - constructors
    - Inheritance
    - Operator overloading
  - Templates
  - STL
  - Memory Management
Logistics

• Final exam
  – Good news…bad news
  – Good news
    • Last day of finals, November 18th
  – Bad news
    • 8am-10am
  – Room
    • 01-3338

Plan for this week

• I/O Week
  – Today: Exceptions
  – Tomorrow: Exceptions II
  – Thursday: Exam 2

Before we start

• Any questions??

When things go wrong

• When a program comes upon a problem it cannot solve locally it can:
  – Terminate the program
    • E.g. assertion
  – Return an “error” value
  – Return a valid value and leave the program in a “bad” state.
    • E.g. IOStreams
  – Call some “error handling” functions

Enter…the exception

• Exceptions allow a method to tell the caller when an error has occurred
  – Many times it is the calling function that knows what to do when an error occurs.
  – Exceptions allow the caller to respond to the error rather than the method itself.
  – Different callers may wish to respond to particular errors differently.

C++ Exceptions

• The idea behind C++ exceptions is very much like Java exceptions.

• Like all things C++, though, C++ exceptions do have their quirks.
Throwing exceptions

• In C++, exceptions are thrown by using the `throw` keyword.
  – Unlike Java, there is not a Throwable class.
  – In C++, any item can be thrown
    • Basic datatypes (int, float, etc.)
    • Class objects
    • Pointers to class objects
    • References to class objects

```cpp
class Stack {
public:
  bool isFull();
  void push();
private:
  int size;
  ...
};

void Stack::push() {
  ...
  if (isFull()) throw size;
}
```

Throwing exceptions

• Like in Java, it is more useful to create a hierarchy of Exception classes.

```cpp
class MathError { }
class Overflow : public MathError { }
class Underflow : public MathError { }
class DivideByZero : public MathError { }
```

Throwing exceptions

• Exception classes are not special.
  – They can contain methods/data like any other class.
    ```cpp
class MathError {
  // ...
  virtual void printMessage() const;
}
```
  – They can also be derived from multiple classes
    ```cpp
class NetfileError : public NertworErr, public FileError { … }
```

Catching Exceptions

• Like in Java, C++ uses a `try/catch` block for catching exceptions.

```cpp
void f() {
  try {
    // call to a method that may throw something
  }
  catch (Overflow) {
    // code that handles an overflow error
    ...
  }
  ...
}
```

Catching Exceptions

• Rules for catching exceptions:
  - `try` // something of type E is thrown
  ```cpp
  try {
    // call to a method that may throw something
  }
  catch (H) { // when is the handler invoked?
    // Handler is invoked if
    1. H is the same type as E
    2. E is derived from H
    3. H and E are pointers and 1 & 2 apply to the things they point to.
    4. H is a reference and 1 & 2 hold for the type H refers to.
  ```
Catching Exceptions

void f() {
    throw MathError();
}

void g() {
    try { f(); }
    catch (MathError E) { E.printMessage(); }
}

Copying will occur

Catching Exceptions

void f() {
    throw Overflow();
}

void g() {
    try { f(); }
    catch (MathError E) { E.printMessage(); }
}

Slicing Will Occur

Catching Exceptions

void f() {
    throw new MathError();
}

void g() {
    try { f(); }
    catch (MathError &E) { E.printMessage();
        delete E; // to prevent a memory leak
    }
}

No Slicing Will Occur

Catching Exceptions

• To catch anything, regardless of type, use the … syntax.

try {
    // something
} catch (...) { 
    // catches anything thrown at you
}

Rethrowing exceptions

• Once caught, an exception can be rethrown by using the throw keyword:

try {
    // something
} catch (...) {
    // catches anything thrown at you
    // and throws it back
    throw;
}
Catching exceptions

• Handlers in a try / catch block are tried in the order in which they appear.

```java
try { // something}
catch (Overflow)
  { // handle overflow}
catch (MathError)
  { // handle any math error}
catch (...)  
  { // handle anything}
```

Catching exceptions

• Erroneous ordering:

```java
try { // something}
catch (...)  
  { // handle anything}
catch (MathError)
  { // it’ll never get here}
catch (Overflow)
  { // or here}
```

Catching exceptions

• Questions

Exception specification

• Like Java, what gets thrown by a method can be declared when defining the function.
• Unlike Java, this declaration is not required.
  – But if there, it is guaranteed to throw only what’s specified.

```java
void f() throw (Overflow, int)
{| 
  if ( ) throw Overflow();
  else throw ?;
|}
```

Exception specification

• If unspecified (default), the function may throw anything:

```java
int f();  // can throw anything
```
• To indicate that a function will never throw an exception

```java
int g() throw ();  // throws nothing
```
Exception specification

• In exception hierarchies, derived classes may only restrict what is thrown

```cpp
class B {
    virtual void f();    // can throw anything
    virtual void g() throw (X, Y);
    virtual void h() throw (X);
};
class C : public B {
    virtual void f() throw (X);      // ok
    virtual void g() throw (X);       // ok
    virtual void h() throw (X,Y);     // not okay
};
```

Exception specification

• If a method tries to throw something NOT in it’s specification
  – Violates its guarantee
  – Program will abort

Exceptions in Java

• When an exception is thrown, the exception gets passed to the calling function.
• This function may:
  – Catch the exception, then perform whatever error handling is appropriate or
  – Pass the exception up the call stack to the function that called it.
• If an exception reaches the main method and is not caught and handled, the program will terminate.

Exceptions in C++

• Same is true in C++ except:
  – Pass the exception up the call stack is implicit
  – No need for function to specify this in definition of function.

Exceptions in C++

• C++ details
  – If an exception reaches the main method and is not caught and handled, the program will call it’s terminate method
    • By default the terminate function is std::terminate() which will call abort() and abort execution.
    • Programmer can redefine the terminate function

Exception specification

• If a method tries to throw something NOT in it’s specification
  – Violates its guarantee
  – Program will actually call the unexpected handler
    • By default the unexpected handler is std::unexpected which will call the terminate function.
    • Can be redefined by programmer.
• Questions?
Stack unwinding

• When an exception is thrown in C++
  – Call stack is searched for first function to catch the data thrown.
    • If none found, program will terminate.
    • If one is found:
      – All local variables from all methods on stack from method that threw the exception to that which caught it, will have it’s destructor called.
      – Note that this is not true for objects allocated on the heap.
      – This is called “stack unwinding”

Stack unwinding

• If an exception is caught and handled
  – Execution continues from next statement after the try/catch block.

Standard Exceptions

• There are some standard exceptions
  – bad_alloc – thrown by new
  – bad_cast – thrown by dynamic_cast
  – bad_typeid – thrown by typeid
• STL exceptions
  – out_of_range
  – invalid_argument
  – overflow_error
  – ios_base::failure

Standard Exceptions

• There is no guarantee or rule that forces one to derive their exceptions from this hierarchy.
Exceptions

- Questions?

- Next time
  - Using exceptions in practice